Minimizing Average Flow-time under Knapsack Constraint

Suman k. Bera* Syamantak Das† Amit Kumar†

April 28, 2015

Abstract

We give the first logarithmic approximation for minimizing average flow-time of jobs in the subset parallel machine setting (also called the restricted assignment setting) under a single knapsack constraint. In a knapsack constraint setting, each job has a profit, and the set of jobs which get scheduled must have a total profit of at least a quantity Π. Our result extends the work of Gupta, Krishnaswamy, Kumar and Segev (APPROX 2009) who considered the special case where the profit of each job is unity. Our algorithm is based on rounding a natural LP relaxation for this problem. In fact, we show that one can use techniques based on iterative rounding.

Keywords: Approximation algorithms, Multiprocessor scheduling, Average flow-time

1 Introduction

Classical scheduling problems that minimize an underlying objective function require that all the jobs in the input get processed. However, for many applications, one might require only a subset of the jobs to be scheduled so as to meet a pre-specified hard profit. In this paper, we consider the well studied objective of minimizing the average flow time of jobs in a multiprocessor environment under such a hard profit constraint which we call the *knapsack* constraint. Formally, there is an associated profit π_j with each job. The goal is to schedule a subset of jobs whose sum of profits is at least a fixed quantity Π. Equivalently, the sum of profit (weights) of the rejected jobs should fit in a knapsack of size B. Note that the special case where each job has unit profit corresponds to the problem where there is a lower bound on the number of jobs to be processed.

Charikar et al. [3] initiated the study of this model where there is a lower bound on the number of jobs which need to be scheduled. Gupta et al. [8] extended this model to a wide variety of scheduling problems. In particular, they consider the problem of minimizing average flow-time on identical parallel machines where jobs have unit profit. In this paper, we generalize this result in two directions as described below.

1.1 Our Results and Techniques

We give the first $O(\log P)$-approximation algorithm for the problem of minimizing average flow-time of jobs in the restricted assignment setting (each job can only be processed by a subset of machines) under a single knapsack constraint(`FlowKnap`). Here P is the ratio of the largest to the

*Dartmouth College, USA. email: suman.k.bera.gr@dartmouth.edu

†Department of Computer Science and Engg., IIT Delhi. email: {sdas, amitk}@cse.iitd.ac.in

smallest job size. Previously, such a result was only known for the parallel machines setting and when each job had unit profit [8].

Our technique is based on a standard LP relaxations for these problems. Directly working with such a relaxation turns out to be involved and tricky. Instead, we show that one can often extract some interesting properties of a fractional solution, and we can write a second simpler LP relaxation based on these properties. This simpler LP relaxation can then be iteratively rounded.

As has been remarked in [8], the presence of knapsack constraint makes the problem significantly harder than the non-profit version even in the simple setting of a *single machine*. In fact, this is the version we shall consider in greater details and later on show how to extend it to the case of subset parallel assignments.

As is standard for algorithms for flow-time, we first divide the jobs into *classes* based on their sizes p_j – a job is of class k if its processing time lies in the range $[2^k, 2^{k+1}]$. Using established techniques, it turns out that it is enough to get a constant factor approximation for the special case when all the jobs are of the same class *provided* our rounding algorithm schedules these jobs in the same timeslots in which class k jobs were scheduled by the fractional solution (upto some constant number of slots violations). Thus, we consider the case when all jobs are of roughly the same size and they are only allowed to be processed in some given region of the timeline. We call this the Scheduling with Forbidden Regions(`ForbidFlow`) problem. We first write a time-indexed LP relaxation for this problem which schedules each job to a fractional extent between 0 and 1. Given a fractional solution, we perform the following steps:

- We create a modified instance and its corresponding fractional solution by compressing all the *gaps* in the LP schedule, i.e., we move the jobs back in time (and decrease their release dates) to fill all available space. This considerably simplifies the subsequent rounding steps. Note that in the algorithm of Gupta et al. [8], considerable technical work is required to get around this problem. The high level reason is that one would want to change the fractional assignment of jobs such that these become 0 or 1. Naive ways to achieve this may end up charging to the entire length of the schedule including the gaps in between. So one needs to do a finer analysis to prevent this. However, once we move the jobs back, this issue goes away.

- It turns out we can get rid of the time-indexed LP relaxation, and just work with the fraction to which a job is processed. We write a simpler LP relaxation to capture this and show that it can be rounded using iterative rounding.

- Finally, we show how to *expand* this schedule to obey the earlier release dates without incurring much in the flow-time.

The analysis essentially shows that we do not incur a lot of cost compared to LP in each of these steps.

1.2 Related Work

Scheduling jobs on machines subject to various constraints is one of the central problems in combinatorial optimization. See [9, 11] for earlier work on the problems of minimizing makespan and `GAP` in multiple machines environment. Garg and Kumar [5, 6] gave upper and lower bounds for minimizing average flow-time of jobs under various settings.

More recently, there has been a growing interest in the study of *scheduling with rejections*. Several works ([1, 4, 2]) have studied the prize collecting model where one has to pay a penalty for

the jobs that are rejected. On the other hand, Charikar et al. [3] considered scheduling under the knapsack constraint where the objective is to schedule jobs so as to achieve a target profit. Guha et al. [7] presented an LP rounding based algorithm for minimizing average completion-time with outliers where the outlier constraint is violated by a constant factor. Gupta et al. [8] gave approximation algorithms for scheduling with outliers under various settings. They give a constant factor approximation for `GAP` under the knapsack constraint(approximation factor improved by Saha and Srinivasan [10]), an $O(\log k)$ approximation for average completion time on unrelated machines minimization under k knapsack constraints and a logarithmic approximation for minimizing total(average) flow time of jobs on identical parallel machines under knapsack constraint where each job has *unit profit*.

2 Preliminaries

In this section, we formally describe the scheduling problems that will be considered in this paper. We will be given a set of jobs J and a set of machines M. In the subset parallel machines setting, each job j specifies a subset S_j of machines on which it can be processed and its processing time on these machines is p_j. Recall that in the *knapsack* constraint, each job j also comes with a profit π_j, and we are given a lower bound Π. Given a release date r_j for each job j, The flow-time of a j is the difference between its completion time and release date. The goal is to schedule a subset of jobs of total profit at least Π while minimizing the total flow-time of scheduled jobs. We allow jobs to be pre-empted but we do not allow migration across machines. However, our approximation ratio holds with respect to a migratory optimum as well.

We need to solve a related problem, which we call scheduling with forbidden regions(`ForbidFlow`). Here, we are given a single machine, and the setting is same as minimizing total flow-time with knapsack constraint. Further, all processing times are within a factor of 2 of each other. We are also given a quantity $z(t)$ for each timeslot $[t, t+1]$. Any schedule can only use $1 - z(t)$ amount of space in this timeslot. In Section 3, we give an $O(1)$-approximation algorithm with an additive $\sum_t z(t)$ factor for this problem. We use this result as a subroutine for the `FlowKnap` problem (Section 4).

3 Scheduling with Forbidden Regions on a Single Machine

In this section, we give an approximation algorithm for the `ForbidFlow` problem. An instance $\mathcal{I} = \texttt{ForbidFlow}(J, Q, \Pi, z)$ is similar to an instance of `FlowKnap` – each job $j \in J$ has associated parameters r_j, p_j and π_j as before while Π is a hard profit. Processing times of the jobs are bounded in a range $[Q, 2Q]$ for some positive integer $Q, Q \geq 1$. Let T be a guess for the time at which the optimal solution completes processing all jobs. Moreover, z is a vector in $[0, 1]^T$. In any time slot $[t, t+1]$, the schedule can do at most $1 - z_t$ amount of processing. For a time slot t we refer to the quantity z_t as *forbidden space* while the rest is called *available space*. We write the following LP relaxation for this problem. Here x_{jt} refers to the amount of processing of job j done during the timeslot $[t, t+1]$. This variable is defined only for $t \geq r_j$. For a job j, y_j refers to the fraction of job j which gets processed. It is well-known that the objective function is a lower bound on the value of the flow-time of the corresponding schedule [5]. The first term in the objective function refers to *fractional flow-time* and the second term refers to half of the processing time of the schedule. Constraint (1) states that a job should be processed to the extent of p_j if y_j is 1. Constraint (2) says that a timeslot $[t, t+1]$ can only be used to the extent of $1 - z_t$. Finally, constraint (3) states

that the total profit of jobs selected for processing must be at least Π. Thus, this is a relaxation for the `ForbidFlow` problem.

$$\min \sum_{j=1}^{n} \sum_{t} \frac{x_{jt}(t - r_j)}{2Q} + \frac{1}{2} \sum_{j} y_j p_j \qquad \texttt{(LP1)}$$

$$\sum_{t \geq r_j} x_{jt} = y_j p_j, \forall j \tag{1}$$

$$\sum_{j=1}^{n} x_{jt} \leq 1 - z_t, \forall t \tag{2}$$

$$\sum_{j=1}^{n} y_j \pi_j \geq \Pi \tag{3}$$

$$0 \leq y_j \leq 1, \forall j \tag{4}$$

Let $(x^\star, y^\star)$ be an optimal solution to `LP1` and $\texttt{flow}(x^\star, y^\star)$ denote the LP objective. We state the main theorem of this section.

Theorem 3.1. *There is a polynomial time algorithm to schedule jobs of total profit at least Π such that total flow-time of the scheduled jobs is at most $O(\texttt{flow}(x^\star, y^\star) + \sum_t z_t)$.*

3.1 The Rounding Algorithm

Our algorithm will only use the $y_j^\star$ variables – this is so because of the following lemma.

Lemma 3.2. *Let $j_1, \ldots, j_n$ be an ordering of the input jobs according to increasing release dates. There is an optimal solution to `LP1` which processes the jobs in the order $j_1, \ldots, j_n$. Hence, the solution does not cause any pre-emption, and can be described by just specifying the y_j values.*

Proof. Consider an optimal solution. For a timeslot $[t, t+1]$, let $u(t)$ denote $\sum_j x_{jt}$, i.e., the total processing done by the solution in this timeslot. It is easy to see that the objective function is equal to

$$\sum_{t} \frac{u(t)t}{2Q} - \sum_{j} \frac{r_j y_j p_j}{2Q} + \frac{1}{2} \sum_{j} y_j p_j.$$

Therefore, if we reorder the jobs processed in this solution in any manner, the value of the objective will remain unchanged as long as we use only $u(t)$ amount of volume in the slot $[t, t+1]$. It is easy to see that processing the jobs in the order $j_1, \ldots, j_n$ where j_i is processed to a fraction of y_{j_i} will respect the release dates as well, and hence, this will be an optimal solution as well.

□

The lemma above implies that, given $y_j^\star$ values, we can deduce the $x_{jt}^\star$ values. We now describe the details of each of the conceptual steps of our algorithm:

Closing the gaps: Given the instance $\mathcal{I}$ and the solution $(x^\star, y^\star)$, our first step is to create a new instance $\mathcal{I}'$ and a corresponding fractional solution $(\bar{x}, \bar{y})$. The idea is that in the solution $(x^\star, y^\star)$ there may be gaps in the fractional schedule – a gap is unused timeslot which appears in the middle of a schedule. The reason why we cannot process a job during the gap is that all subsequent jobs

CloseGaps :

1. Initialize $\bar{y}_j = 0$ for all j and $\bar{x}_{jt} = 0$ for all j, t.
2. For $l = 1, 2, \ldots, n$
   Let t_l be the first time such that $z(t_l) + \sum_{s=1}^{l-1} \bar{x}_{j_s t_l} < 1$
   (i.e., there is some more processing which can be done at this time slot)
   In the instance $\mathcal{I}'$, define $\bar{s}_{j_l} = t_l$.
   Process the job j_l to an extent of $y^\star_{j_l}$ from time t_l onwards, i.e.,
   for $t = t_l$, set $\bar{x}_{jt} = \min(1 - z_t - \sum_{s=1}^{l-1} \bar{x}_{j_s t}, p_j y^\star_j)$ and
   for $t > t_l$, iteratively set $\bar{x}_{jt} = \max(0, \min(1 - z_t, p_j y^\star_j - \sum_{t'<t} \bar{x}_{jt}))$.
   Set $\bar{y}_{j_l} = y^\star_{j_l}, \bar{r}_j = r_j - (s^\star_j - \bar{s}_j)$.

Figure 1: Algorithm for modifying $\mathcal{I}$ to $\mathcal{I}'$

have release dates beyond this timeslot. In $\mathcal{I}'$ we would like to move jobs to the left so that these gaps go away. $\mathcal{I}'$ will be identical to $\mathcal{I}$ except that the release date of a job in $\mathcal{I}'$ will appear before that in $\mathcal{I}$.

We give some notation first. Let $j_1, \ldots, j_n$ be an ordering of the input jobs according to increasing release dates. For a job j, let $s^\star_j$, the starting time of j, denote the first timeslot $[t, t+1]$ in which the solution $(x^\star, y^\star)$ processes j, i.e., the smallest t such that $x^\star_{jt} > 0$ – we can assume that $y^\star_j > 0$ for all jobs j because we may not consider any job for which $y^\star_j = 0$. When we construct $\mathcal{I}'$, $\bar{r}_j$ will denote the release date of job j in this instance, and $\bar{s}_j$ will denote the first timeslot $[t, t+1]$ for which $\bar{x}_{jt} > 0$. We iteratively modify the instance $\mathcal{I}$ to $\mathcal{I}'$ and construct a fractional schedule simultaneously. The procedure is described in Fig. 1. It essentially moves each job back so that all gaps get filled. The starting time of the job moves back accordingly. The release dates also move back by the same amount as the starting time. It is not difficult to see that the relative ordering of the release dates in $\mathcal{I}'$ are same as that in $\mathcal{I}$.

Since we are only moving the processing of a job back in time, it is easy to prove by induction that for each job j, $\bar{s}_j \leq s^\star_j$ and so, $\bar{r}_j \leq r_j$. For indices $t_1 < t_2$, let $\texttt{avail}(t_1, t_2)$ denote the total available space in the timeslots $[t_1, t_1 + 1], \ldots, [t_2, t_2 + 1]$, i.e., $\sum_{t=t_1}^{t_2} (1 - z_t)$. Let $\texttt{forbid}(t_1, t_2)$ be the total forbidden space in the corresponding timeslots, i.e., $\sum_{t=t_1}^{t_2} z_t$. Clearly, $\texttt{avail}(t_1, t_2) + \texttt{forbid}(t_1, t_2) = t_2 - t_1 + 1$.

We note one simple fact which will be useful later. Again, the reason why this is true is because in $(x^\star, y^\star)$, we could have had gaps between the execution of two jobs j and j', but these will not be present in $(\bar{x}, \bar{y})$.

Claim 3.3. *Let j, j' be jobs with $r_j \leq r_{j'}$. Then $\texttt{avail}[s^\star_j, s^\star_{j'}] \geq \texttt{avail}[\bar{s}_j, \bar{s}_{j'}]$.*

The first observation is that the cost of the solution $(\bar{x}, \bar{y})$ is close to that of $(x^\star, y^\star)$.

Lemma 3.4.

$$\sum_{j,t} \frac{\bar{x}_{jt}(t - \bar{r}_j)}{2Q} \leq \sum_{j,t} \frac{x^\star_{jt}(t - r_j)}{2Q} + \sum_t z_t + \sum_j y^\star_j p_j.$$

Proof. Let $\bar{C}_j$ be the completion time of j in the schedule $(\bar{x}, \bar{y})$. For a job j, we will show that

$$\sum_{t \geq \bar{s}_j} \frac{\bar{x}_{jt}(t - \bar{s}_j)}{2Q} \leq \sum_{t \geq s^\star_j} \frac{x^\star_{jt}(t - s^\star_j)}{2Q} + \sum_{t=\bar{s}_j+1}^{\bar{C}_j} z_t + y^\star_j p_j \tag{5}$$

For the sake of argument, we divide the processing of job j into very small pieces such that each piece "fits" in one timeslot. More formally, suppose ε is a small enough positive quantity such that all positive $x^\star_{jt}$ and $\bar{x}_{jt}$ values are integral multiples of ε. We can think of $N_j = \frac{y_j p_j}{\varepsilon}$ such contiguous pieces of j, which get processed in both schedules in this order. Now, such a piece c getting processed at time t in $x^\star$ will contribute $\frac{\varepsilon(t-r_j)}{2Q}$ to the sum $\sum_{t \geq s^\star_j} \frac{x^\star_{jt}(t-s^\star_j)}{2Q}$. We claim that this piece will finish processing in the schedule $(\bar{x}, \bar{y})$ by time $t' = \bar{s}_j + 1 + (t - s^\star_j) + \texttt{forbid}(\bar{s}_j + 1, \bar{C}_j)$. Indeed, if $t' \geq \bar{C}_j$, then there is nothing to prove because by definition of $\bar{C}_j$, j will finish by $\bar{C}_j \leq t'$. So assume $t' < \bar{C}_j$. Now, $\texttt{forbid}(\bar{s}_j + 1, t')$ is at most $\texttt{forbid}(\bar{s}_j + 1, \bar{C}_j)$. So,

$$\texttt{avail}(\bar{s}_j + 1, t') \geq t' - \bar{s}_j - \texttt{forbid}(\bar{s}_j + 1, \bar{C}_j) \geq t - s^\star_j + 1.$$

But then the piece c should finish processing by time t' in $(\bar{x}, \bar{y})$ because the total processing requirement of pieces of j coming before c (including c) is at most $t - s^\star_j + 1$. Thus, this piece will contribute at most $\frac{\varepsilon(1 + (t-s^\star_j) + \texttt{forbid}(\bar{s}_j+1, \bar{C}_j))}{2Q}$ to the sum $\sum_{t \geq \bar{s}_j} \frac{\bar{x}_{jt}(t-\bar{s}_j)}{2Q}$. Summing over all the pieces of j, we get

$$\begin{aligned}
\sum_{t \geq \bar{s}_j} \frac{\bar{x}_{jt}(t - \bar{s}_j)}{2Q} &\leq \sum_{t \geq s^\star_j} \frac{x^\star_{jt}(1 + (t - s^\star_j) + \texttt{forbid}(\bar{s}_j + 1, \bar{C}_j))}{2Q} \\
&\leq \sum_{t \geq s^\star_j} \frac{x^\star_{jt}(t - s^\star_j)}{2Q} + y^\star_j p_j + \texttt{forbid}(\bar{s}_j + 1, \bar{C}_j),
\end{aligned}$$

because $p_j \geq 1$ and $\sum_t x^\star_{jt} = y^\star_j p_j \leq 2 y^\star_j Q$. This proves inequality (5). Summing this over all jobs and noting that the closed intervals $[\bar{s}_j + 1, \bar{C}_j]$ are disjoint for different jobs, we get

$$\sum_{j,t} \frac{\bar{x}_{jt}(t - \bar{s}_j)}{2Q} \leq \sum_{j,t} \frac{x^\star_{jt}(t - s^\star_j)}{2Q} + \sum_t z_t + \sum_j y^\star_j p_j. \tag{6}$$

This implies the desired result because

$$\begin{aligned}
&\sum_{j,t} \frac{\bar{x}_{jt}(t - \bar{r}_j)}{2Q} - \sum_{j,t} \frac{x^\star_{jt}(t - r_j)}{2Q} \\
&= \sum_{j,t} \frac{\bar{x}_{jt}(t - \bar{s}_j)}{2Q} + \sum_{j,t} \frac{\bar{x}_{jt}(\bar{s}_j - \bar{r}_j)}{2Q} - \sum_{j,t} \frac{x^\star_{jt}(t - r_j)}{2Q} - \sum_{j,t} \frac{x^\star_{jt}(r_j - s^\star_j)}{2Q} \\
&= \sum_{j,t} \frac{\bar{x}_{jt}(t - \bar{s}_j)}{2Q} - \sum_{j,t} \frac{x^\star_{jt}(t - s^\star_j)}{2Q}
\end{aligned}$$

because $r_j - s^\star_j = \bar{r}_j - \bar{s}_j$ and $\sum_{j,t} \bar{x}_{j,t} = \bar{y}_j = y^\star_j = \sum_{j,t} x^\star_{j,t}$.

□

Iterative Rounding: Now we show how to round the solution $(\bar{x}, \bar{y})$ for the instance $\mathcal{I}'$. Again, our rounding algorithm will only use the $\bar{y}$ values. We achieve this by an iterative rounding procedure which works with a simple LP relaxation – this simpler LP relaxation only looks at the $\bar{y}_j$ values. We first motivate the new LP relaxation. It is not hard to see that the objective function for the solution $(\bar{x}, \bar{y})$ can be written as $\sum_t \frac{tu(t)}{2Q} - \sum_j \frac{\bar{r}_j \bar{y}_j p_j}{2Q} + \frac{1}{2}\sum_j p_j \bar{y}_j$, where $u(t)$ is the amount of processing done in timeslot $[t, t+1]$. It will turn out that the quantity $u(t)$ will not change. Hence, we can treat the objective function as $-\sum_j \frac{\bar{r}_j \bar{y}_j p_j}{2Q} + \frac{1}{2}\sum_j p_j \bar{y}_j$.

Our algorithm will iteratively reject or select some jobs. Hence, at any point of time during our algorithm, we will work with a residual budget Π' – this is the remaining profit we need to recover from jobs for which we have not made a decision. Our algorithm will also maintain a set of jobs J' initialized to the set of all jobs. These will be the set of jobs about which our algorithm has not decided whether they will be scheduled or not. For a set of jobs J', let $F(J')$ be the first three jobs in J' (according to release dates).

Given jobs j, j' we say that $j \prec j'$ if $\bar{r}_j \leq \bar{r}_{j'}$. For a job j, let $\bar{V}_j$ denote the amount of processing done by the solution $(\bar{x}, \bar{y})$ on jobs released before j (including j), i.e., $\bar{V}_j = \sum_{j':j' \prec j} \sum_t \bar{x}_{j't} = \sum_{j':j' \prec j} \bar{y}_{j'} p_{j'}$. Our new LP relaxation is shown below.

$$\min -\sum_{j \in J'} \frac{p_j y_j \bar{r}_j}{2Q} + \frac{1}{2}\sum_{j \in J'} y_j p_j \qquad \text{(LP2)}$$

$$\sum_{j \in J'} y_j \pi_j = \Pi' \tag{7}$$

$$\sum_{j' \in J' : j \prec j'} y_{j'} p_{j'} = V_j \qquad \forall j \in J' - F(J') \tag{8}$$

$$0 \leq y_j \leq 1 \qquad \forall j \in J' \tag{9}$$

Constraint (7) ensures the total profit remains unchanged. Constraints (8) imposes that total volume of scheduled jobs which come before j remain restricted to a certain volume V_j (which will be initially $\bar{V}_j$). We write these constraints for $j \in J' - F(J')$.

IterRound:

1. Initialize $J' \leftarrow J$, $V_j \leftarrow \bar{V}_j$ for all jobs j. Set $S = \emptyset$, $\Pi' = \Pi$.
2. While $|J'| > 3$
   (i) Find a vertex solution y to LP2.
   (ii) If there is a job $j \in F(J')$ with $y_j = 0$, set $J' \leftarrow J' \setminus \{j\}$.
   (iii) Else if there is a job $j \in F(J')$ with $y_j = 1$, set $S \leftarrow S \cup \{j\}$ and
      a. $J' \leftarrow J' \setminus \{j\}$.
      b. $\Pi' \leftarrow \Pi' - \pi_j$.
      c. $\forall j', j \prec j', j' \in J'$, $V_{j'} \leftarrow V_{j'} - p_j$.
3. $S \leftarrow S \cup J'$
4. Return S.

Figure 2: Algorithm for rounding LP relaxation for $\mathcal{I}'$

Now we present the iterative rounding algorithm for converting the fractional schedule into an

integral one (Algorithm **IterRound**). Note that the LP relaxation does not write the constraints (8) for the first three jobs in J'. This will ensure that a vertex solution will have at least one variable which is 0 or 1. In the end, we shall return the jobs which were added to the set S and the remaining three jobs in J'. This is the set of jobs chosen by our algorithm. We will prove that either step 2(ii) or step 2(iii) will be executed in each iteration of this algorithm. Let S be the set of jobs returned by the above algorithm. Finally, we show how to schedule these jobs feasibly with respect to the original release dates.

We first show that the iterative rounding algorithm will terminate.

Lemma 3.5. *Consider a vertex solution y to* `(LP2)`*. Assuming $|J'| \geq 4$, there exists a job $j \in F(J')$ for which y_j is 0 or 1.*

Proof. Suppose not. Let j_4 be the fourth job (according to release date) in J'. Note that constraints (8) is written for j_4 and jobs released after j_4. By subtracting the constraint for all jobs j' released after j_4 from the constraint for job j_4, we get an equivalent LP. However, in this LP, the variables $y_{j_1}, y_{j_2}, y_{j_3}$, where j_1, j_2, j_3 are the first three jobs in J', appear in only two tight constraints – constraint (8) for j_4 and constraint (7).

□

We give some more notations. We index the iterations of the `while` loop in the algorithm starting from n downwards. Thus, at the beginning of iteration k, $|J'| = k$. Let `LP2(`k`)` denote the corresponding LP, and $y^{(k)}$ be the vertex solution found by the Algorithm **IterRound** for this LP. So the first iteration finds the solution $y^{(n)}$, then $y^{(n-1)}$ and so on till $y^{(4)}$. Let $y^{(3)}$ be a vertex solution to `LP2(3)` when $|J'|$ is 3 (and V_j, Π' values are given by the end of iteration indexed 4) – even though our algorithm will not use this LP solution, it will be useful for analysis. Again, let $V_j^{(k)}$ and $\Pi^{(k)}$ to be the values of V_j and Π' when $|J'| = k$. Let $S^{(k)}$ be the set S when $|J'| = k$. So, $S^{(n)}$ is $\emptyset$.

We now give a procedure which given values y_j for all jobs $j \in J$, outputs a corresponding schedule x_{jt}. This is similar to the algorithm in Figure 1. The procedure arranges the jobs in ascending order of release dates and schedules them in this order. The procedure is described in Figure 3. Note that the schedule itself does not care about the release dates of the jobs and so may not even respect the release dates.

Given the solution $y^{(k)}$ for a subset of jobs J', we extend it to the set of all jobs in J by setting $y_j^{(k)} = 1$ for all $j \in S^{(k)}$, and 0 for jobs in $J - J' - S^{(k)}$. We shall call this the *extended solution* $y^{(k)}$. Let $x^{(k)}$ be the corresponding schedule of these jobs given by calling **ScheduleJobs** on the extended solution $y^{(k)}$. We shall use $\texttt{flow}(x^{(k)}, y^{(k)})$ to denote the objective function value if we view this as a solution to `(LP1)` with respect to release dates $\bar{r}$, i.e.,

$$\texttt{flow}(x^{(k)}, y^{(k)}) = \sum_j \sum_t \frac{x_{jt}^{(k)}(t - \bar{r}_j)}{2Q} + \frac{1}{2}\sum_j y_j^{(k)} p_j.$$

Lemma 3.6. *The extended solution $y^{(k)}$ is feasible to* `(LP2)` *during every iteration k of the algorithm* **IterRound**. *Further, each of these solutions processes jobs of total volume $\sum_j \bar{y}_j p_j$, and*

$$\texttt{flow}(x^{(3)}, y^{(3)}) \leq \texttt{flow}(\bar{x}, \bar{y}).$$

ScheduleJobs :

Input : Values $y_j \in [0,1]$ for all jobs $j \in J$.
Output : A schedule of the jobs $\bar{x}_{jt}$, such that $\sum_t \bar{x}_{jt} = y_j$.

1. For $l = 1, 2, \ldots, n$
   Let t_l be the first time such that $z(t_l) + \sum_{s=1}^{l-1} \bar{x}_{j_s t_l} < 1$
   (i.e., there is some more processing which can be done at this time slot)
   Process the job j_l to an extent of y_{j_l} from time t_l onwards, i.e.,
   for $t = t_l$, set $\bar{x}_{j_l t} = 1 - z_t - \sum_{s=1}^{l-1} \bar{x}_{j_s t_l}$ and
   for $t > t_l$, iteratively set $\bar{x}_{j_l t} = \max(0, \min(1 - z_t, p_{j_l} y^\star_{j_l} - \sum_{t'<t} \bar{x}_{j_l t'}))$.

Figure 3: Algorithm for building schedule from a solution y

Proof. Clearly, $\bar{y}$ is a feasible solution to `LP2(`n`)`. Hence, `LP2(`n`)` has non-empty set of feasible solutions, and $y^{(n)}$ is well-defined. Assume (by induction) that `LP2(`k`)` has a non-empty set of feasible solutions, and hence $y^{(k)}$ is well-defined. Given the solution $y^{(k)}$, suppose we select a job j with $y_j^{(k)} = 1$ in iteration k (the other case is similar). It is easy to see that $y^{(k)}$ is also a feasible solution to `LP2(`$k-1$`)` in the next iteration if we do not consider the variables corresponding to j. Further, the constraint (8) for the last job (according to release date) ensures that the total volume of the processed jobs in the solution $y^{(k-1)}$ is same as that in the solution $(x^{(k)}, y^{(k)})$ minus p_j. Hence, if we consider the extended solution $y^{(k-1)}$ to all the jobs, then the volume processed by it does not change.

Since all the solutions $y^{(k)}$ process the same amount of volume, and do not have any gaps, they will occupy each slot to the same extent. So

$$\texttt{flow}(x^{(k)}, y^{(k)}) = \sum_t \frac{u(t)t}{2Q} - \sum_j \frac{\bar{r}_j y_j^{(k)} p_j}{2Q} + \frac{1}{2} \sum_j y_j^{(k)} p_j.$$

The quantity $u(t)$ denotes the amount of processing done in $[t, t+1]$, and will not depend on k. Since `LP2` treats the other terms above in the objective, it is easy to show that $\texttt{flow}(x^{(k-1)}, y^{(k-1)}) \leq \texttt{flow}(x^{(k)}, y^{(k)})$. □

We now consider the schedule corresponding to S. Define a solution $\tilde{y}$ as $\tilde{y}_j = 1$ iff $j \in S$. Let $(\tilde{x}, \tilde{y})$ be the corresponding schedule obtained by calling **ScheduleJobs** on $\tilde{y}$.

Lemma 3.7. *For the schedule* $(\tilde{x}, \tilde{y})$*,*

$$\texttt{flow}(\tilde{x}, \tilde{y}) \leq \texttt{flow}(\bar{x}, \bar{y}) + 12 \sum_j \bar{y}_j p_j + 6 \sum_t z_t.$$

Proof. The schedule for S is obtained from $(x^{(3)}, y^{(3)})$ by adding 3 jobs in the beginning. This will require *shifting* the jobs scheduled in $(x^{(3)}, y^{(3)})$ to the right by at most $6Q$ amount of available space so that these 3 jobs can be accommodated. The processing time of these 3 jobs is at most

$6Q$. Now, if we look at any timeslot $[t, t+1]$, the number of alive jobs at this time can increase by at most 6 (because these many new jobs which were being processed before t could now be getting processed after t). Hence the increase in flow-time is at most 6 times the makespan of the schedule, which is at most 6 times $\sum_t z_t$ plus the total processing volume. Assuming that there are at least 4 jobs in optimal solution (otherwise we could just enumerate), this would mean that the processing time of the 3 new jobs can be charged to the processing volume of $(x^{(3)}, y^{(3)})$. This proves the desired result.

□

Corollary 3.8. *The total profit of the jobs S returned by the algorithm* **IterRound** *is at least Π.*

Proof. We prove by induction on k that the total profit of the jobs in $S - S^{(k)}$ is at least $\Pi^{(k)}$. Base case is $k = 3$. We know that there is a feasible solution to `(LP2)` when there are just 3 jobs remaining in J' (Lemma 3.6). In this fractional solution, we get a profit of at least $\Pi^{(3)}$. Our algorithm picks all the three jobs and hence its profit must be at least $\Pi^{(3)}$ as well. Assuming that this is true for some k, it is easy to see that the statement holds true for $k-1$ as well (we are updating $\Pi^{(k)}$ accordingly). When $k = n$, $S^{(k)}$ is $\emptyset$, and so we are done.

□

Lemma 3.9. *For a job $j \in J$, let $p(S_{\prec j})$ be the total processing time of jobs in S which are released before j. Then $p(S_{\prec j})$ lies in the interval $[\bar{V}_j - 6Q, \bar{V}_j + 6Q]$.*

Proof. Consider the first iteration of `LP2` when j is among the first 3 jobs of J' – say this is iteration k. So far, we have chosen a set of jobs $S^{(k)}$. Since we are always writing the constraint (8) for j, we know that $V_j^{(k)} = \bar{V}_j - p(S^{(k)})$. Since there is a feasible solution to this LP (Lemma 3.6), we know that $V_j^{(k)} \geq 0$, and so the jobs selected in S have processing time at least $\bar{V}_j - V_j^{(k)}$. But now $V_j^{(k)}$ involves just 3 jobs, and so it cannot be larger than $6Q$. This proves the lower bound on $p(S_{\prec j})$. The upper bound follows similarly – the only eligible jobs in $p(S_{\prec j})$ are either those in $S^{(k)}$ or the first three jobs of this iteration.

□

Corollary 3.10. *Let j, j' be jobs in J, $j \prec j'$. Let $S_{j,j'}$ be the of jobs in S which lie between j and j' (excluding j and j') with respect to the order $\prec$. Then $p(S_{j,j'}) \leq 12Q + \sum_{j'' : j \prec j'' \prec j'} p_{j''} y^\star_{j''}$.*

Using the above lemma, we now show that it is possible to modify the solution $(\tilde{x}, \tilde{y})$ such that it obeys the release dates $\bar{r}_j$ for the jobs. But we will not need such a result in our analysis because we can directly use the above result. We now give proofs for the final schedule constructed by our algorithm.

Opening the gaps: Recall that $(\tilde{x}, \tilde{y})$ is the schedule obtained for S where we just start from the beginning and fill available space without looking at the release dates (Figure 3). As a final step in our algorithm, we convert this into a feasible schedule that respects all release dates. We consider these jobs in the ascending order of release dates r_j (which is same as the ordering with respect to $\bar{r}$). We schedule j in the earliest available slots after r_j.

Let $\hat{C}_j$ be the completion time of job $j \in S$ in the final schedule. We need to account for $\sum_{j \in S} (\hat{C}_j - r^\star_j)$. We split this sum into two parts and give bounds on them separately.

Lemma 3.11.

$$\sum_{j \in S} (s^\star_j - r^\star_j) \leq 8\texttt{flow}(\tilde{x}, \tilde{y}) + 7 \sum_t z_t.$$

Proof. Note that $s^\star_j - r^\star_j = \bar{s}_j - \bar{r}_j$. Thus, it is enough to consider the sum $\sum_{j\in S}(\bar{s}_j - \bar{r}_j)$. Fix a job j. Let $\tilde{s}_j$ be the first slot in which it gets processed in $(\tilde{x}, \tilde{y})$. Clearly, the cost of $(\tilde{x}, \tilde{y})$ is at least $\sum_{j\in S}(\tilde{s}_j - \bar{r}_j)$, because all of job j gets processed after time $\tilde{s}_j$ in this solution, and so, even the fractional flow-time is at least $\tilde{s}_j - \bar{r}_j$. It remains to consider $\sum_{j\in S}(\bar{s}_j - \tilde{s}_j)$. We estimate this sum as follows. Fix a time t during the schedule. We bound how many intervals $[\tilde{s}_j, \bar{s}_j]$ can contain t. Let $\tilde{J}$ be the set of jobs in S for which the interval $[\tilde{s}_j, \bar{s}_j]$ contains t. Suppose, for the sake of contradiction, that $|\tilde{J}| \geq 7$. Since the available space between the starting time of any two consecutive jobs in $(\tilde{x}, \tilde{y})$ is at least Q (because it needs to process one job), the available space between $\tilde{s}_{j_f}$ and $\bar{s}_{j_f}$ is at least $7Q$. But then the difference between $\bar{V}_{j_f}$ and $p(S_{\prec j_f})$ is at least $7Q$, which contradicts Lemma 3.9. So, $\sum_{j\in S}(\bar{s}_j - \tilde{s}_j)$ is at most 7 times the makespan of $(\tilde{x}, \tilde{y})$, which is at most 7 times $\texttt{flow}(\tilde{x}, \tilde{y}) + \sum_t z(t)$.

□

Lemma 3.12.

$$\sum_{j\in S}(\hat{C}_j - s^\star_j) \leq 14\texttt{flow}(\tilde{x}, \tilde{y}) + 14\sum_t z_t.$$

Lemma 3.12. Consider the intervals $[s^\star_j, \hat{C}_j]$. We claim that a fixed time t can be contained in at most a constant number of such intervals. Let us see why. Let $\tilde{J}$ be the jobs for which the corresponding interval contains t. Let j_l be the last job in this set. Now, let j_u be the first job such that there is no gap during the execution of j_u till j_l in the final schedule. So, all of processing of jobs in the range $j_u, \ldots, j_l$ is done after j_u. Also, this sequence contains all the jobs in $\tilde{J}$ (because at time t they are all alive). Now note that the processing time of jobs in this set is at least $\texttt{avail}[s^\star_{j_u}, s^\star_{j_l}] + p(\tilde{J}) - 2Q$ – indeed, the jobs in the range $j_u, \ldots, j_l$ get processed from $r^\star_{j_u} \leq s^\star_{j_u}$ till $\hat{C}_{j_l}$, $\texttt{avail}[s^\star_{j_u}, \hat{C}_{j_l}] \geq \texttt{avail}[s^\star_{j_u}, s^\star_{j_l}] + \texttt{avail}[t, \hat{C}_{j_l}]$, and we process at least $|\tilde{J}| - 1$ jobs in the latter interval. Claim 3.3 and Corollary 3.10 now imply that the total processing time of the jobs in $j_u, \ldots, j_l$ is at most $12Q + \texttt{avail}[s^\star_{j_u}, s^\star_{j_l}]$. This implies that $p(\tilde{J}) \leq 14Q$, and hence $|\tilde{J}| \leq 14$. This proves the desired result.

□

Combining Lemma 3.11, Lemma 3.12, Lemma 3.7 and Lemma 3.4, we get Theorem 3.1.

4 Flow Time Minimization on Multiple Machines Under Knapsack Constraint

We use our algorithm for `ForbidFlow` to design an algorithm for `Flowknap` in the subset parallel setting. Recall that each job has a release date r_j, a processing requirement p_j and a profit π_j while a target profit Π has to be attained by the scheduled jobs. The machines are identical, but a job can go only to a subset of machines S_j. The objective is to minimize total flow time of the scheduled jobs. The proof of the following theorem can be found in full version of the paper.

Theorem 4.1. *There is a polynomial time $O(\log P)$-approximation algorithm for* `Flowknap` *in the subset parallel setting under a single knapsack constraint. Here, P is the ratio between largest and the smallest processing time of a job.*

We write a natural LP relaxation for this problem, which is an extension of the LP relaxation used in [6]. We divide jobs into classes – a job is of class k if its processing time lies in the interval $[2^{k-1}, 2^k)$. Our algorithm runs over several iterations – in each iteration it schedules jobs of a

particular class, say class k. Using the optimal solution to the LP relaxation, we figure out the extent to which each timeslot processes jobs of class k. We now use the `ForbidFlow` algorithm to schedule jobs of class k where we are allowed to use a timeslot to this extent only. Combining the solutions for all classes gives Theorem 4.1.

LP Relaxation We write the following LP relaxation for this problem. It is identical to the LP relaxation used in [6], with the additional constraint (12) which states the knapsack constraint. Also note that the quantity T is a guess for a suitable upper bound for the time at which the optimal solution completes processing of all jobs. In the LP relaxation below, $\widetilde{p_j}$ refers to the smallest power of 2 which is at least p_j.

$$\min \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{t=0}^{T}\left(\frac{x_{ijt}}{\widetilde{p_j}}\left(t+\frac{1}{2}-r_j\right)+\frac{x_{ijt}}{2}\right) \qquad (\texttt{LPFlowKnap})$$

subject to

$$p_j y_j = \sum_{i=1}^{m}\sum_{t=0}^{T} x_{ijt}, \text{ for all } j \tag{10}$$

$$\sum_{j=1}^{n} x_{ijt} \leq 1, \text{ for all } t, i \tag{11}$$

$$\sum_{j=1}^{n} y_j w_j \geq \Pi \tag{12}$$

$$x_{ijt} = 0, \text{ if } r_j < t \text{ or } i \notin S_j \tag{13}$$

$$0 \leq x_{ijt}, y_j \leq 1, \text{ for all } j, t \tag{14}$$

The following fact is well-known(see e.g., [5]).

Lemma 4.2. *The above LP is a valid relaxation for* `Flowknap` *in the subset parallel machines setting with a knapsack constraint.*

As is standard for such problems, we first divide the jobs into various classes $C_1, C_2, \ldots C_L$, where $L = \log P$. A job j belongs to class C_k if $p_j \in [2^{k-1}, 2^k)$. Clearly for all jobs j of the same class, $\tilde{p_j}$ is same.

Rounding Algorithm for Flow Time With Knapsack Constraints: Let $(x^\star, y^\star)$ be an optimal solution to `LPFlowKnap`. Our algorithm proceeds in L iterations. We introduce some notations first. Denote the solution restricted to jobs in C_k as $(x_k^\star, y_k^\star)$. At the kth iteration, we start with $(x_k^\star, y_k^\star)$ and work with the jobs in C_k. Note that the fractional solution may schedule portions of a job $j \in C_k$ on different machines in parallel up to a total extent of $y_j p_j$. We rearrange them so that the entire $y_j p_j$ volume is processed on only one machine in S_j. This is carried out by invoking the unsplittable flow formulation deployed in [6] – we refer to this as the `GK` algorithm. This results in different machines processing disjoint sets of jobs in the new solution. At this point, we use `ForbidFlow` as a subroutine on each of the machines. For each slot, forbidden regions are defined to be regions that are occupied by jobs that do not belong to C_k. The total target profit is set to be the fractional profit which comes from C_k in the solution $(x_k^\star, y_k^\star)$. Fig. 4 formalizes the description.

Analysis

kth Class:

Input: Solution $(x_k^\star, y_k^\star)$
Output: Solution (x^k, y^k) and a schedule for selected jobs in C_k

1. Initialize $x^k \leftarrow x_k^\star, y^k \leftarrow y_k^\star$.
2. Run `GK` algorithm on (x^k, y^k). Let the new solution be (x', y').
3. For each machine i do
   For time $t = 0, 1, \ldots, T$
   $z_i(t) = \sum_{j \in C_{k'}, k' \neq k} x'_{ijt}$
   Set $J \leftarrow \{j : j \in C_k, \sum_t x'_{ijt} > 0\}$, $Q \leftarrow 2^k$, $\Pi \leftarrow \sum_{j \in J} y'_j \pi_j$
   Let $(\widetilde{x}, \widetilde{y})$ be a solution to `ForbidFlow`(J, Q, Π, z_i)
   For $t = 0, 1, \ldots, T$ and each $j \in J$
   Set $x^k_{ijt} \leftarrow \widetilde{x}_{ijt}$, $y^k_j \leftarrow \widetilde{y}$.

Figure 4: kth iteration of the rounding algorithm for `FlowKnap`

We prove the following lemma which in turn will lead to a proof of Theorem 4.1

Lemma 4.3. *$flow(x^k, y^k) \leqslant \mathcal{O}(flow(x_k^\star, y_k^\star) + U)$ where U is the total processing time of all jobs in the system. Further, total profit of scheduled jobs is at least $\sum_{j \in C_k} y_j^\star \pi_j$*

Proof. The increase in LP value due to `GK` algorithm can be accounted for by using the following Claim 4.3 from [6].

Claim 4.4. *$flow(x', y') \leqslant flow(x_k^\star, y_k^\star) + U$*

However it should be noted that we use a slightly modified version of the `GK` where demand of each job is $y_j p_j$. It is not hard to see that the Claim 4.4 extends to this modified instance as well. Now consider the `ForbidFlow` step in the above algorithm for a particular machine i. It is easy to see that (x', y') forms a feasible solution to the LP relaxation of the `ForbidFlow` instance. Recall that by virtue of `GK` algorithm, each machine processes disjoint sets of jobs in solution (x', y'). Denote $Z_i^k = \sum_t z_i(t)$ for the kth iteration. Then from Theorem 3.1 and the above claim, for jobs restricted to those being processed on i, $flow(x^k, y^k) \leqslant \mathcal{O}(flow(x_k^\star, y_k^\star) + Z_i^k)$. We sum up over all machines to obtain:

$$flow(x^k, y^k) \leqslant \mathcal{O}(flow(x_k^\star, y_k^\star) + \sum_i Z_i^k) \leqslant \mathcal{O}(flow(x_{k-1}^\star, y_{k-1}^\star) + U)$$

Further, from Section 3, we know that `ForbidFlow` schedules enough jobs to meet the profit requirement of B on a particular machine. Again summing over all machines proves the lemma. □

Proof. (Theorem 4.1) Recall that we run Algorithm **kth Class** for each of the L classes. Denote the final solution by (x^f, y^f). Using Lemma 4.3 and adding up for components of each class, we have

$$flow(x^f, y^f) \leqslant \sum_{k=1}^{L} \mathcal{O}(flow(x_k^\star, y_k^\star) + U) \leqslant \mathcal{O}(flow(x^\star, y^\star) + L \cdot U)$$

Arguments similar to [6] show that integral flow-time for the final schedule S_f is upper bounded by $flow(x^f, y^f) + \mathcal{O}(L \cdot U)$. Combining all above, flow time of S_f is at most $\mathcal{O}(flow(x^\star, y^\star) + L \cdot U)$. Using lower bounds $flow(x^\star, y^\star)$ and U for OPT, we have flow time of S_f to be at most $\mathcal{O}(1 + \log P)OPT$.

□

Acknowledgements

The second author would like to thank Tata Consultancy Services Research Scholar Program for supporting the work.

References

[1] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Kedar Dhamdhere. Scheduling for flow-time with admission control. In *Proc. ESA, 2003.*

[2] Yair Bartal, Stefano Leonardi, Alberto Marchetti-Spaccamela, Jiří Sgall, and Leen Stougie. Multiprocessor scheduling with rejection. In *Proc. ACM-SIAM SODA, 1996.*

[3] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *Proc. ACM-SIAM SODA, 2006.*

[4] Daniel W. Engels, David R. Karger, Stavros G. Kolliopoulos, Sudipta Sengupta, R. N. Uma, and Joel Wein. Techniques for scheduling with rejection. In *Proc. ESA, 1998.*

[5] Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *Proc. ICALP - Volume Part I, 2006.*

[6] Naveen Garg and Amit Kumar. Minimizing average flow-time: Upper and lower bounds. In *Proc. IEEE FOCS, 2007.*

[7] Sudipto Guha and Kamesh Munagala. Model-driven optimization using adaptive probes. In *Proc. ACM-SIAM SODA, 2007.*

[8] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In *Proc. APPROX/RANDOM, 2009.*

[9] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 1990.

[10] Barna Saha and Arvind Srinivasan. A new approximation technique for resource-allocation problems. In *Proc. ICS, 2010.*

[11] David B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *Proc. ACM-SIAM SODA, 1993.*